Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра компьютерных систем в управлении и проектировании (КСУП)

SQL ОПЕРАТОРЫ ДЛЯ БД СУБД PGADMIN 4 В ПРИЛОЖЕНИИ MICROSOFT VISUAL STUDIO

Отчет по лабораторной работе по дисциплине «Основы разработки баз данных»

Студент гр. 573-3:

_____ Р.В. Слиньков
дата

_____
подпись

Руководитель:

Преподаватель:

_____ _____ Р. О. Остапенко
оценка подпись

_____
дата

Томск, 2025

Введение

В современном мире базы данных играют ключевую роль в системах хранения и обработки информации. Они используются в самых разных областях — от небольших веб-приложений до крупных корпоративных систем. Эффективная организация данных позволяет оптимизировать работу программных продуктов, обеспечить целостность и безопасность информации, а также упростить её поиск и анализ.

Цель данной лабораторной работы:

Изучение основных особенностей формирования запросов в приложении Microsoft Visual Studio для своей предметной области (на примере базы данных для кафе).

Задачи:

Реализовать программу согласно требованиям:

1. Создать форму для работы с SQL-запросами различного характера;
2. Реализовать запросы на выборки:
   - Выборка из одной таблицы;
   - Выборка с условиями;
   - Выборка из нескольких таблиц;
   - Выборку с агрегирующими (статическими) функциями;
   - Полная запись оператора SELECT(предложения SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY);
   - запросы с параметрами с проверкой корректности. Параметры разных типов данных.
3. Использовать подзапросы;
4. Реализовать SQL-запросы с использованием операторов изменения данных:
   - INSERT;
   - UPDATE;
   - DELETE;

- Параметрические запросы с проверкой корректности. Параметры разных типов данных.

5. Протестировать работу полученного приложения.

1 ОСНОВНАЯ ЧАСТЬ

1.1 Описание архитектуры для создания приложения и ход выполнения работы

Windows Forms — это технология для создания настольных приложений на платформе .NET, которая предоставляет мощный инструментарий для разработки пользовательского интерфейса. Она позволяет использовать широкий набор элементов управления, таких как Label для отображения текста, ***TextBox*** для ввода данных, ***ComboBox*** для выбора из списка, ***CheckBox*** для выбора опций, ***PictureBox*** для работы с изображениями, ***DataGridView*** для отображения данных в табличном виде и ***BindingNavigator*** для навигации по записям. Эти элементы обеспечивают гибкость и удобство при создании интерактивных приложений.

В рамках предыдущих лабораторных работ была реализована архитектура приложения, основанная на принципах разделения ответственности и модульности. Для взаимодействия с базой данных использовался **паттерн репозитория**, который представляет собой прослойку между бизнес-логикой приложения и уровнем доступа к данным. Этот паттерн реализован через интерфейсы (***IWarehouseRepository***, ***ISupplierRepository***, ***ISupplyRepository***, ***IStorageZoneRepository***, ***IEmployeeRepository***) и их конкретные реализации в проекте Infrastructure. Каждый репозиторий предоставляет методы для выполнения операций CRUD (Create, Read, Update, Delete) над соответствующей сущностью, например, GetAll, Add, Update, Delete.

А также было реализованы в полной мере связи 1:M и решены связи M:M через добавление промежуточной таблицы и связи 1:M – M:1.

Ход выполнения заданий

1. **Создание формы для работы с SQL-запросами различного характера:**

В рамках лабораторной работы была создана новая форма ***SQLQueries*** в приложении Windows Forms, предназначенная для демонстрации работы с различными SQL-запросами. Эта форма не является центральной частью приложения, а служит дополнительным инструментом для выполнения запросов к базе данных. Форма была организована с использованием трех вкладок (**TabPages**):

- **Вкладка «Запросы на выборку»**: На этой вкладке реализованы различные виды SELECT-запросов, включая выборку из одной таблицы, выборку с условиями, выборку из нескольких таблиц, выборку с агрегирующими функциями и полную форму ***SELECT*** с предложениями ***WHERE***, ***GROUP BY***, ***HAVING***, ***ORDER BY***. Добавлены кнопки для выполнения запросов и поля для ввода параметров.
- **Вкладка «Подзапросы»**: Здесь реализованы коррелированные и некоррелированные подзапросы с использованием соответствующих методов репозитория. Для ввода параметров добавлены поля, а результаты отображаются в таблице.
- **Вкладка «Изменение данных»**: На этой вкладке настроены операции ***INSERT***, ***UPDATE*** и ***DELETE***. Использованы радиокнопки для выбора типа операции и поля ввода для параметров, таких как ID записи, дата учёта, количество и другие.

Форма интегрирована с репозиторием ***SQLQueriesRepository*** для выполнения запросов, а результаты отображаются в ***DataGridView***.

2. **Реализация подзапросов:**

Для работы с подзапросами использовалась существующая архитектура приложения. Шаблон "Одиночка" (***Singleton***), реализованный ранее в классе ***AppContext***, обеспечил централизованный доступ к репозиторию ***SQLQueriesRepository***. Были настроены методы для выполнения

коррелированных и некоррелированных подзапросов, а также добавлена валидация входных данных (например, проверка корректности ID).

3. **Реализация SQL-запросов с использованием операторов изменения:**

На вкладке "Изменение данных" реализованы операции ***INSERT***, ***UPDATE*** и ***DELETE***. Добавлена валидация вводимых данных: проверка формата даты, корректности числовых значений и существования записей. Обработка ошибок настроена с выводом сообщений пользователю (см. Приложение А, Листинг 1.1 и 1.2).

1.2 Результаты выполнения

В результате выполнения лабораторной работы была добавлена новая форма ***SQLQueries*** в Windows Forms приложение "Система складского учёта". Эта форма не является основной частью приложения, а представляет собой дополнительный инструмент для работы с SQL-запросами, демонстрирующий выполнение различных типов запросов к базе данных. Реализованные функции подробно описаны ниже:

- **Создание формы SQLQueries для работы с SQL-запросами**:
  - Форма ***SQLQueries*** была разработана как вспомогательный модуль приложения, предоставляющий пользователю возможность выполнять SQL-запросы через интуитивно понятный интерфейс. Она организована в виде трёх вкладок, каждая из которых отвечает за определённый тип операций:
    - **Вкладка «Запросы на выборку»**: Позволяет выполнять широкий спектр **SELECT**-запросов. Простая выборка из одной таблицы (рисунок 1.1) отображает все записи, включая такие поля, как ID записи, дата учёта, количество, ID сотрудника, ID поставки и ID зоны хранения (ПРИЛОЖЕНИЕ А, Листинг 1.1 — SQL-запрос для простой выборки из таблицы ***AccountingRecords***, реализованный в методе ***GetSimpleProductAccountingRecords*** репозитория.). Выборка с условиями (рисунок 1.2) реализована с использованием параметров, например, выбор записей по ID сотрудника, где пользователь вводит ID в текстовое поле, и в ***DataGridView*** отображаются только соответствующие записи (ПРИЛОЖЕНИЕ А, Листинг 1.2 — SQL-запрос для выборки с условием по ID сотрудника, реализованный в методе ***GetRecordsByEmployee*** репозитория.). Выборка из нескольких таблиц (рисунок 1.3), например, объединение таблиц ***AccountingRecords***, ***Employees*** и ***StorageZones***, позволяет отображать связанные данные, такие как имя сотрудника и название зоны хранения (ПРИЛОЖЕНИЕ А, Листинг 1.3 — SQL-запрос для выборки из нескольких таблиц с использованием ***JOIN***, реализованный в методе ***GetAllRecords*** репозитория). Выборка с

агрегирующими функциями (рисунок 1.4) включает расчёт, например, общего количества (***SUM***) или среднего количества (***AVG***) продукции по заданным критериям, с учётом минимального количества записей и начальной даты, указанных пользователем (ПРИЛОЖЕНИЕ А, Листинг 1.4 — SQL-запрос для агрегирующей выборки с использованием ***GROUP BY*** и ***HAVING***, реализованный в методе ***GetAggregateRecords*** репозитория). Полная форма SELECT поддерживает все предложения (SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY), что позволяет, например, сортировать записи по дате учёта или группировать их по зонам хранения с фильтрацией по количеству.

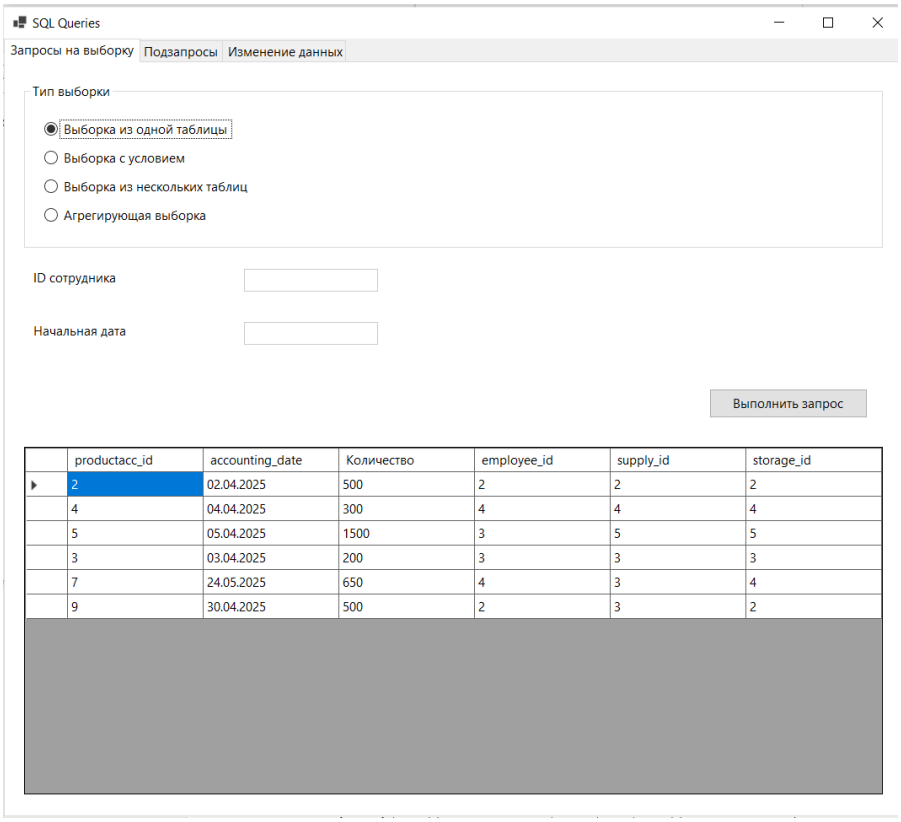

Рисунок 1.1 – Выборка из одной таблицы

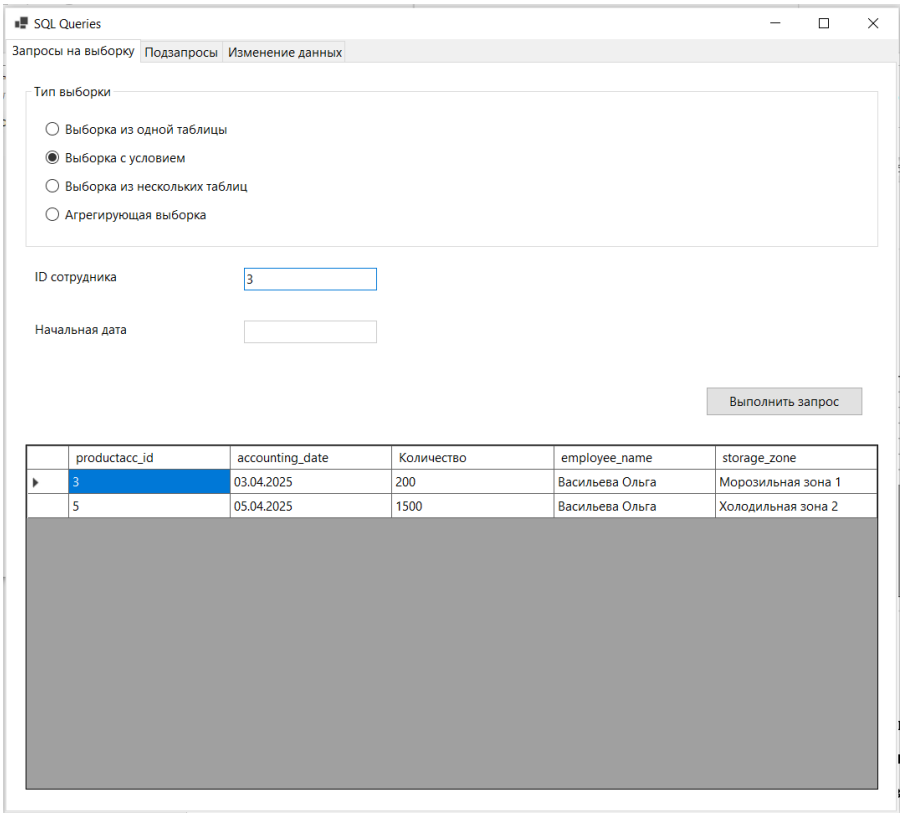

Рисунок 1.2 – Выборка с условием

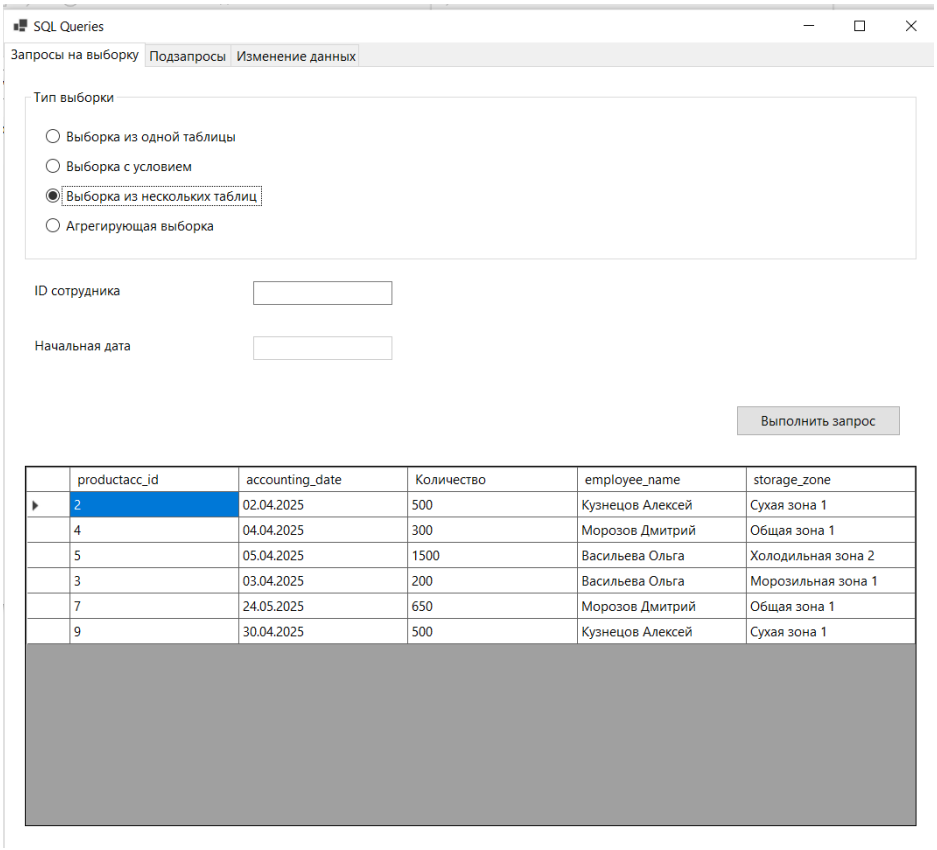

Рисунок 1.3 – Выборка из нескольких таблиц

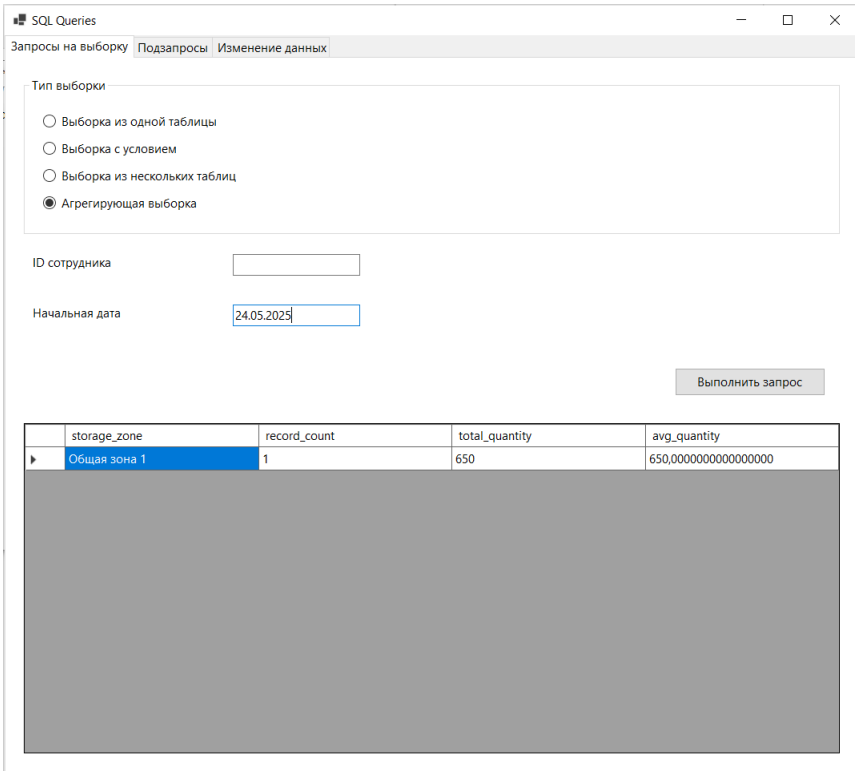

Рисунок 1.4 – Агрегирующая выборка

▪ **Вкладка «Подзапросы»**: Реализованы коррелированные и некоррелированные подзапросы. Коррелированный подзапрос (рисунок 1.5), например, выбирает записи учёта, где ID поставки связан с определёнными условиями в другой таблице, причём внутренний запрос выполняется для каждой строки внешнего запроса (ПРИЛОЖЕНИЕ А, Листинг 1.5 — SQL-запрос для коррелированного подзапроса, реализованный в методе ***GetCorrelatedSubquery*** репозитория). Некоррелированный подзапрос (рисунок 1.6) выполняется независимо, например, выбор всех записей учёта, где ID поставки присутствует в списке поставок за определённый период (ПРИЛОЖЕНИЕ А, Листинг 1.6 — SQL-запрос для некоррелированного подзапроса с использованием ***IN***, реализованный в методе ***GetNonCorrelatedSubquery*** репозитория). Пользователь вводит параметр (ID поставки) в текстовое поле, после чего результаты отображаются в ***DataGridView***. Если данные отсутствуют или введён некорректный ID, пользователю показывается сообщение "Записей не найдено".

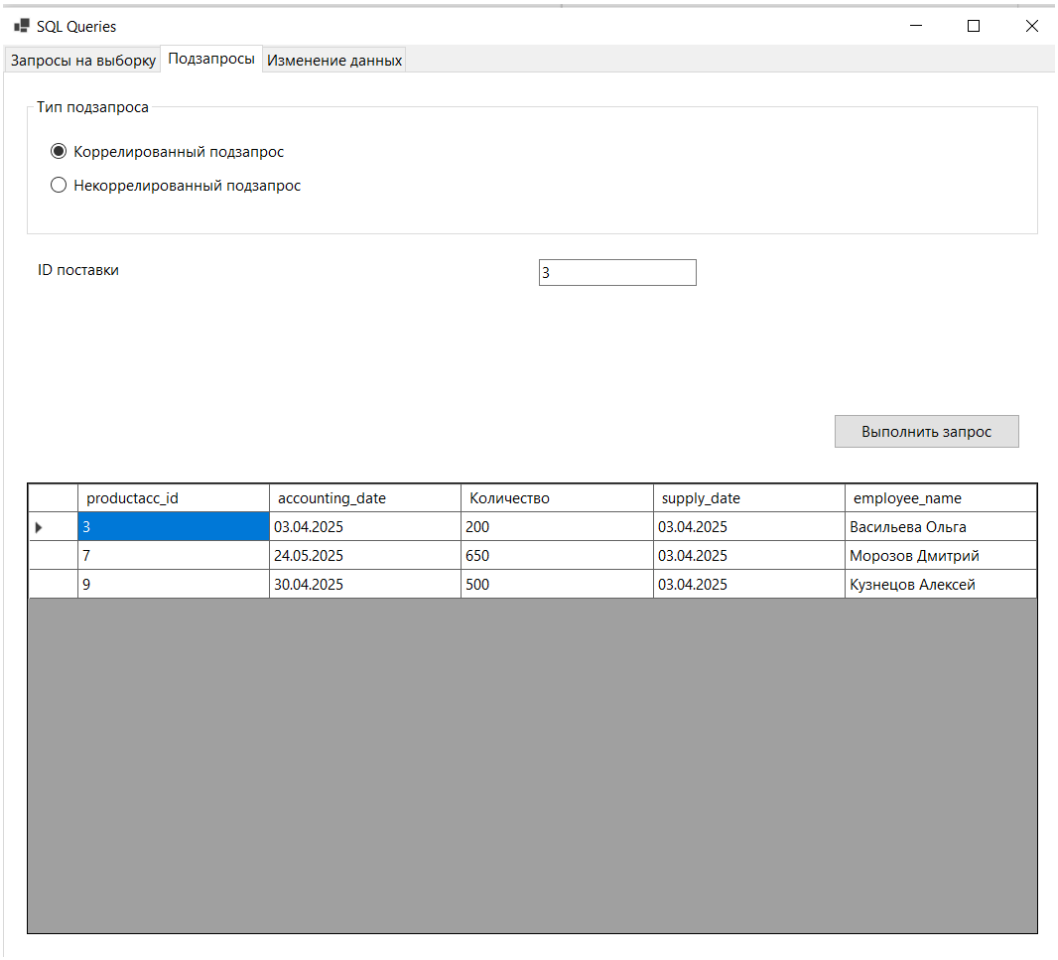

Рисунок 1.5 – Коррелированный подзапрос

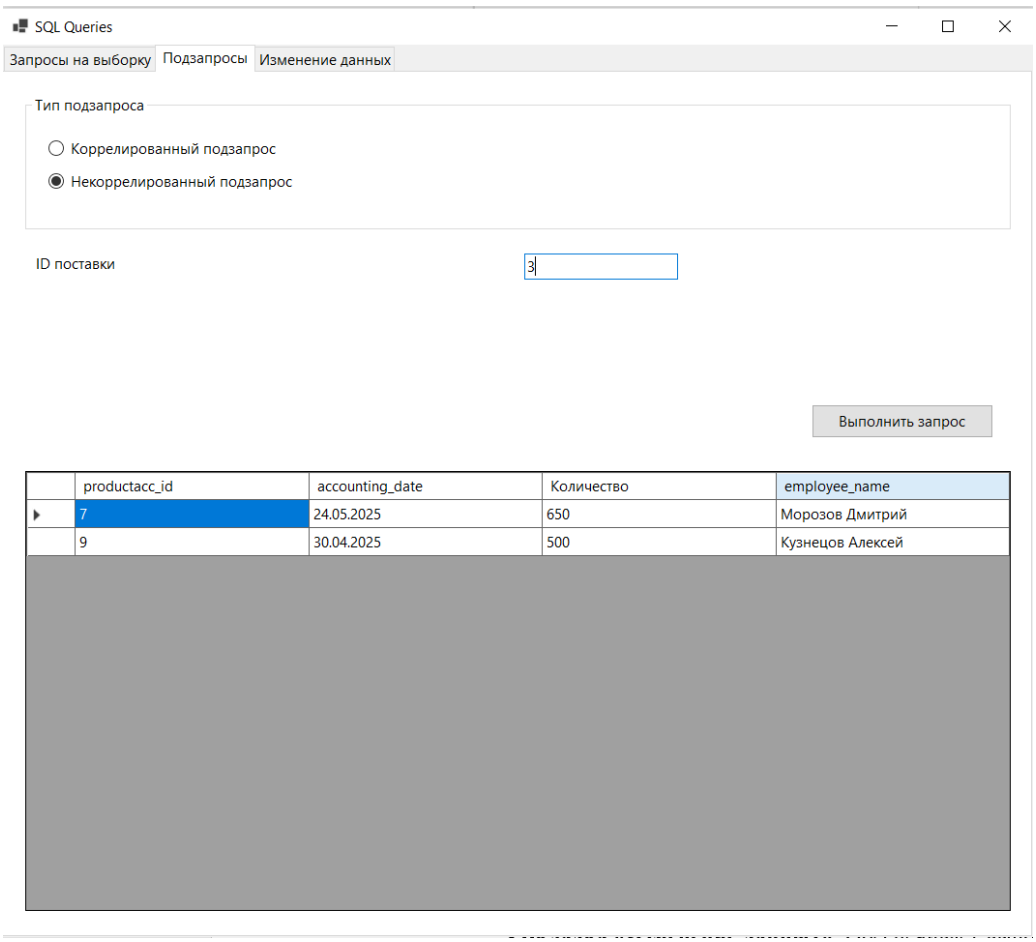

Рисунок 1.6 – Некоррелированный подзапрос

▪ **Вкладка «Изменение данных»**: Обеспечивает выполнение операций ***INSERT***, ***UPDATE*** и ***DELETE***. Для выбора типа операции используются радиокнопки: "Добавить запись учёта" (рисунок 1.7 – 1.8), "Изменить запись учёта" (рисунок 1.9 – 1.10) и "Удалить запись учёта" (рисунок 1.11 – 1.12). При выборе операции "Добавить" пользователь заполняет поля: дата учёта (в формате "ДД.ММ.ГГГГ"), количество (положительное целое число), ID поставки и ID зоны хранения. Например, ввод "25.05.2025", "50", "1", "2" добавляет новую запись в таблицу ***AccountingRecords*** (ПРИЛОЖЕНИЕ А, Листинг 1.7 — SQL-запрос для операции ***INSERT***, реализованный в методе ***InsertRecord*** репозитория). Операция "Изменить" требует указания ID записи и позволяет обновить любые из полей, например, изменить количество с 50 на 75 для записи с ID 1 (ПРИЛОЖЕНИЕ А, Листинг 1.8 — SQL-запрос для операции ***UPDATE***,

реализованный в методе ***UpdateRecord*** репозитория). Операция "Удалить" требует только ID записи и удаляет её из базы, например, запись с ID 1 (ПРИЛОЖЕНИЕ А, Листинг 1.9 — SQL-запрос для операции ***DELETE***, реализованный в методе ***DeleteRecord*** репозитория). Все запросы являются параметрическими, с использованием SQL-параметров (например, ***@AccountingDate***, ***@Quantity***), что защищает от SQL-инъекций. После выполнения операции данные в ***DataGridView*** обновляются автоматически, отображая актуальное состояние таблицы.

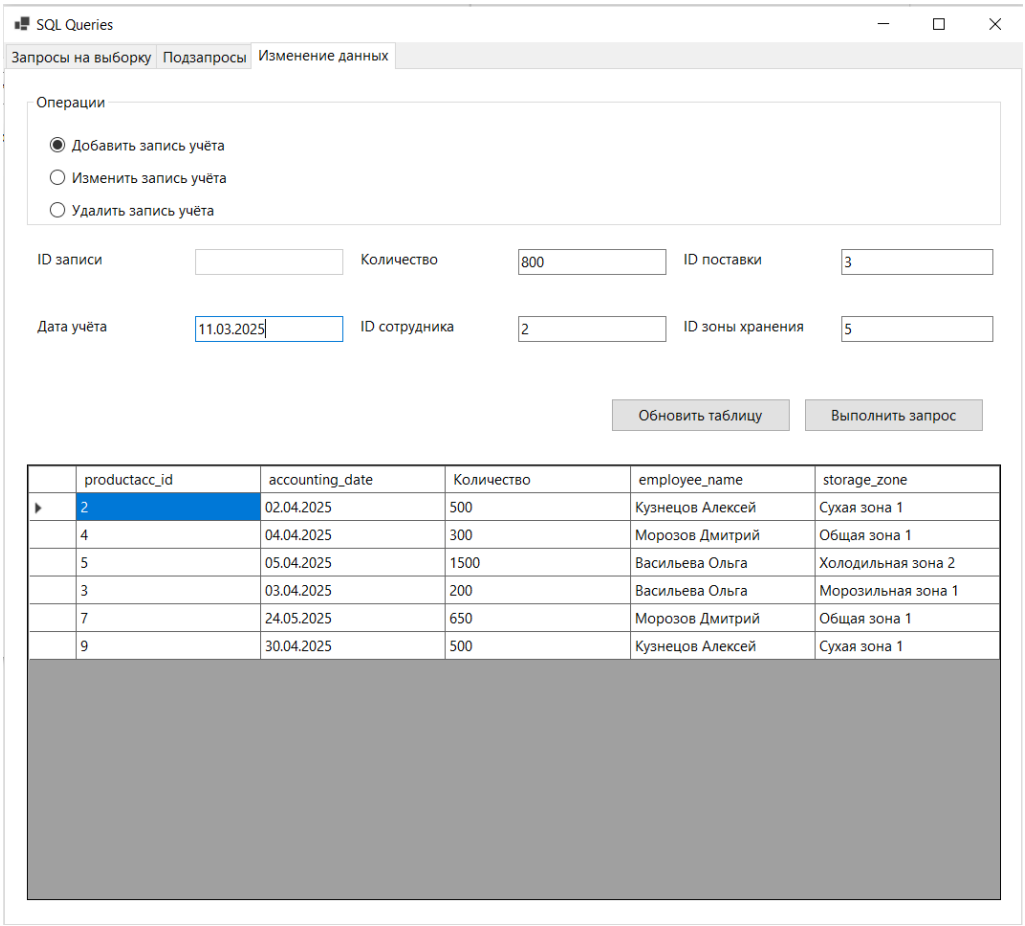

Рисунок 1.7 – «Добавить запись учета» заполнение формы

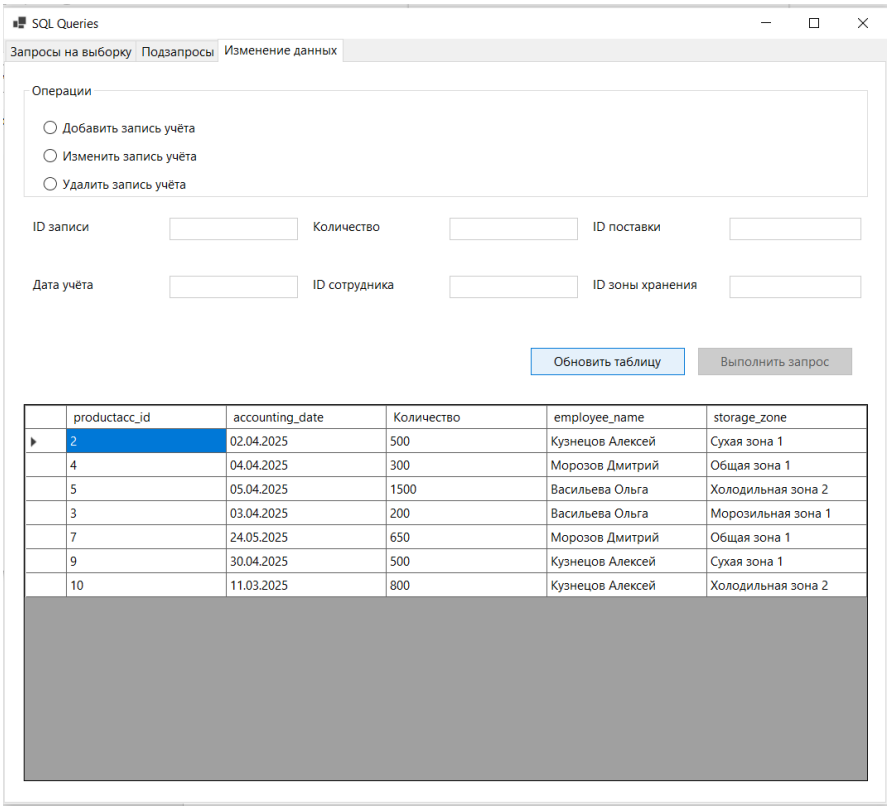

Рисунок 1.8 – «Добавить запись учета» результат работы

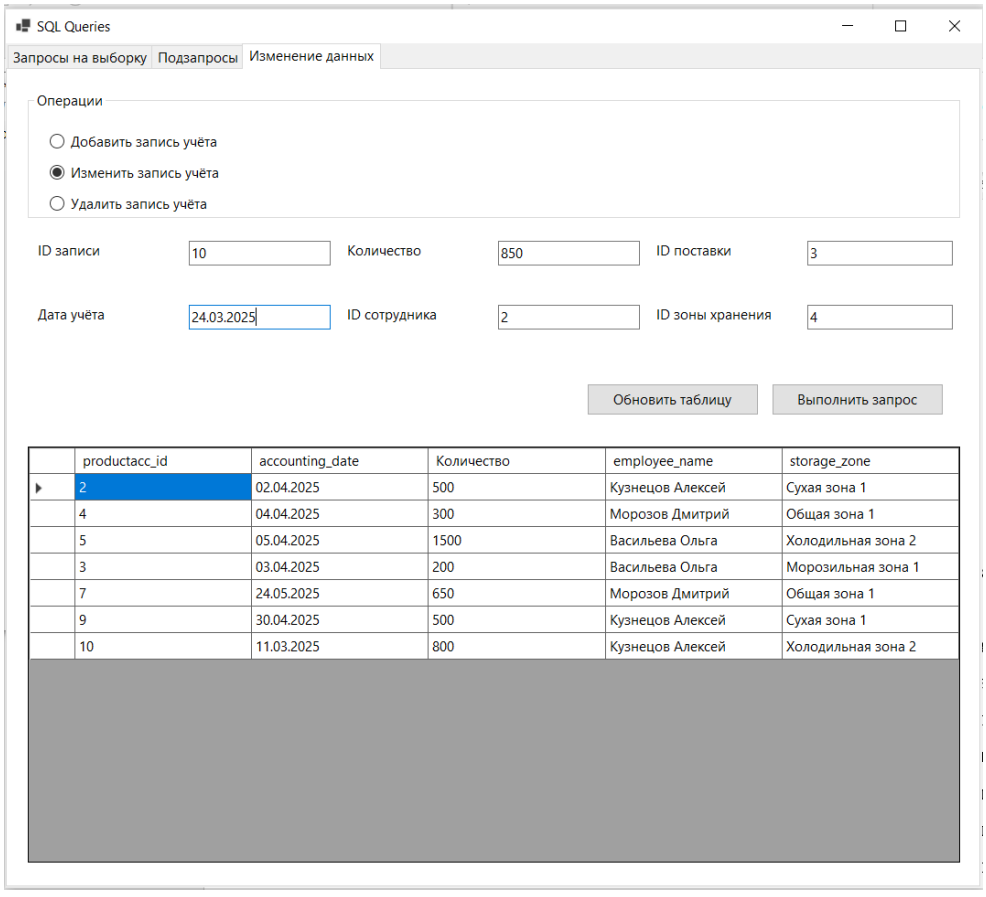

Рисунок 1.9 – «Изменить запись учета» заполнение формы

	10	24.03.2025	850	Кузнецов Алексей	Общая зона 1

Рисунок 1.10 – «Изменить запись учета» результат работы

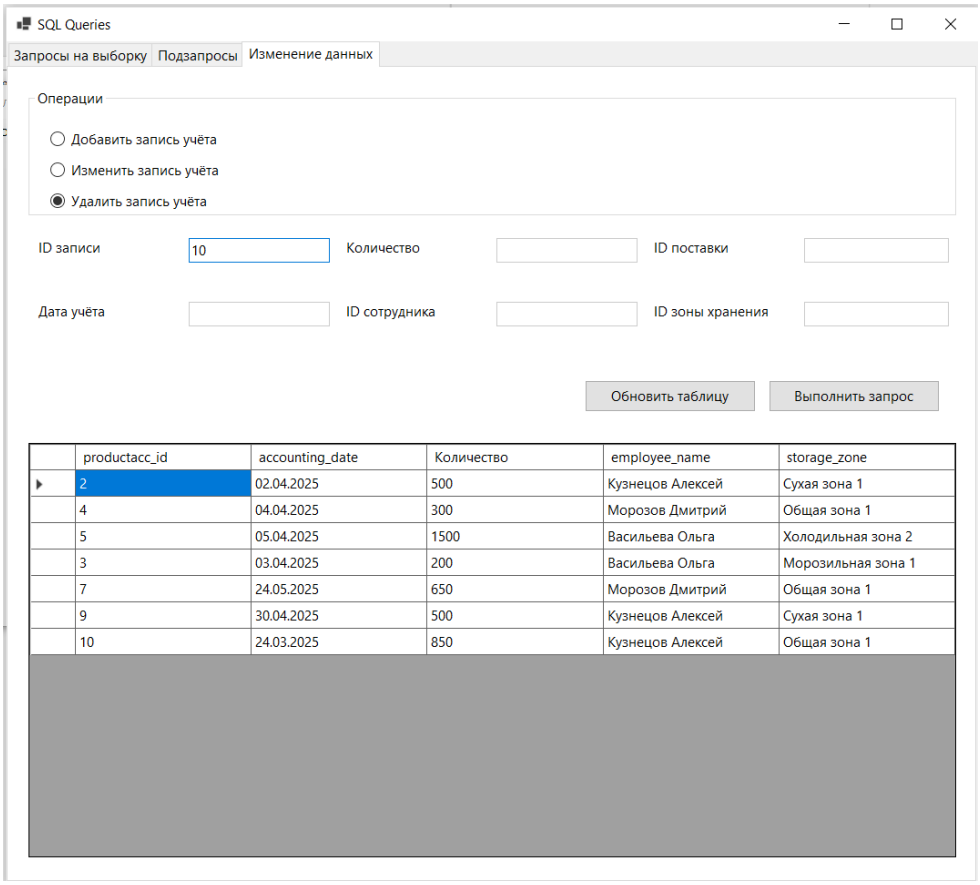

Рисунок 1.11 – «Удалить запись учета» заполнение формы

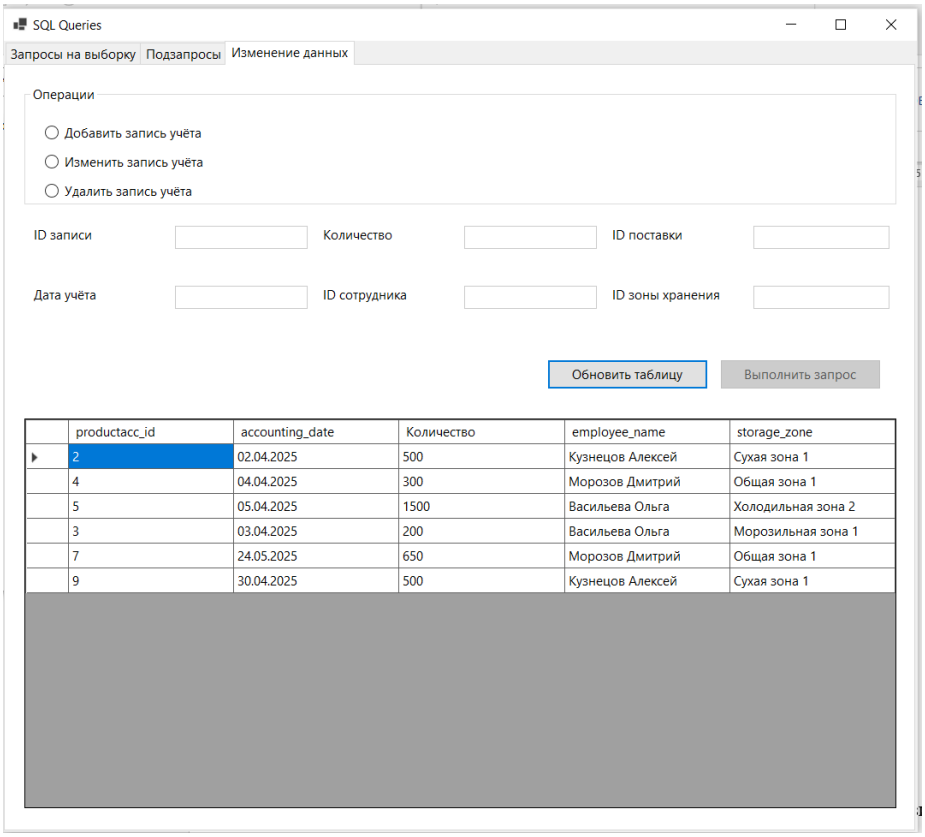

Рисунок 1.12 – «Удалить запись учета» результат работы

- **Реализация подзапросов**:

  - Подзапросы реализованы с использованием репозитория ***SQLQueriesRepository***, доступ к которому осуществляется через класс ***AppContext*** с паттерном ***Singleton***. Коррелированные подзапросы позволяют, например, выбрать записи учёта, где ID поставки связан с поставками, выполненными за последний месяц, причём внутренний запрос выполняется для каждой строки внешнего запроса. Некоррелированные подзапросы, такие как выбор записей с ID поставки из списка поставок определённого поставщика, выполняются независимо. Пользователь вводит ID поставки в текстовое поле, например, "3", и в ***DataGridView*** отображаются только те записи, которые соответствуют условиям подзапроса. При некорректном вводе (например, отрицательное значение ID) или отсутствии данных отображается сообщение "Введите корректный ID поставки" или "Записей не найдено", что делает взаимодействие с приложением информативным и безопасным.

- **Реализация SQL-запросов с использованием операторов изменения**:

  - Операции ***INSERT***, ***UPDATE*** и ***DELETE*** реализованы с учётом всех требований задания. Для добавления записи (***INSERT***) пользователь вводит параметры, такие как дата учёта ("25.05.2025"), количество (например, "100"), ID поставки ("2") и ID зоны хранения ("1"). Запрос выполняется через метод репозитория ***InsertRecord***, который использует параметрический SQL-запрос для вставки данных в таблицу ***AccountingRecords***. Для изменения записи (***UPDATE***) пользователь указывает ID записи (например, "1") и новые значения полей, например, изменяет количество на "150" или дату на "26.05.2025". Метод ***UpdateRecord*** обновляет запись, поддерживая частичное обновление (если поле не заполнено, оно не изменяется). Для удаления записи (***DELETE***) достаточно указать ID записи (например, "1"), после чего метод ***DeleteRecord*** удаляет запись из базы. Валидация данных включает проверку формата даты (например, "ДД.ММ.ГГГГ"), положительности числовых значений и существования записи перед изменением или удалением. Например, попытка удалить несуществующую запись с ID 999 приводит к сообщению "Запись с ID 999 не существует". После каждой операции DataGridView обновляется, показывая актуальные данные.

- **Обработка исключений**:

  - Приложение включает обработку исключений для повышения надёжности. Например, при вводе некорректной даты (например, "25/05/2025" вместо "25.05.2025") или отрицательного количества (например, "-10") пользователю отображается сообщение "Введите корректную дату в формате ДД.ММ.ГГГГ" или "Введите положительное целое число" (Приложение А, Листинг 1.10 — Код валидации входных данных в методе ***ValidateDate***, показывающий обработку некорректного ввода). При попытке изменить или удалить несуществующую запись (например, с ID 999) показывается

сообщение "Запись с ID 999 не существует". Эти меры улучшают пользовательский опыт, предотвращая некорректные операции.

Таким образом, форма ***SQLQueries*** успешно интегрирована в приложение, дополняя его функциональность и демонстрируя возможности работы с различными SQL-запросами в рамках системы складского учёта.

Заключение

В ходе выполнения лабораторной работы №4 была успешно добавлена новая форма ***SQLQueries*** в Windows Forms приложение "Система складского учёта". Форма служит вспомогательным модулем, демонстрирующим работу с различными типами SQL-запросов, и полностью соответствует требованиям лабораторной работы. Приложение предоставляет удобный пользовательский интерфейс для выполнения запросов на выборку, подзапросов и операций изменения данных, обеспечивая гибкость и надёжность при работе с базой данных.

Ключевые результаты работы:

- Реализованы все пункты лабораторной работы, включая создание формы ***SQLQueries*** с тремя вкладками ("Запросы на выборку", "Подзапросы", "Изменение данных"), настройку обработки исключений, использование элементов управления (***Label***, ***TextBox***, ***RadioButton***, ***Button***, ***DataGridView***), а также тестирование приложения. Форма позволяет выполнять **SELECT**-запросы (простые, с условиями, из нескольких таблиц, с агрегирующими функциями), подзапросы (коррелированные и некоррелированные), а также операции ***INSERT***, ***UPDATE*** и ***DELETE*** с параметрами и валидацией данных.
- Архитектура приложения, основанная на паттерне репозитория, обеспечила чёткое разделение бизнес-логики и уровня доступа к данным, что повысило модульность и упростило тестирование. Паттерн "Одиночка" (Singleton), реализованный в классе ***AppContext***, продолжил обеспечивать централизованное управление зависимостями, что упростило доступ к репозиторию ***SQLQueriesRepository*** и поддержание согласованности данных.
- Добавлена обработка исключений для всех операций, включая проверку корректности ввода (например, формата даты и числовых значений) и обработку ошибок при выполнении запросов (например, попытка удаления несуществующей записи). Это повысило надёжность приложения и улучшило пользовательский опыт.

Приложение демонстрирует высокую степень модульности и гибкости благодаря разделению на проекты (***Domain***, ***Infrastructure***, ***UI***, ***Application***). Использование интерфейсов и паттернов проектирования позволяет легко расширять функциональность в будущем, например, добавлять новые типы запросов, интегрировать дополнительные формы для работы с другими сущностями или адаптировать приложение для работы с другими технологиями баз данных.

ПРИЛОЖЕНИЕ А

Листинг 1.1 – SQL-запрос для простой выборки

```
public DataTable GetSimpleProductAccountingRecords()
{
    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        string query = @"
            SELECT productAcc_id, accounting_date, quantity,
employee_id, supply_id, storage_id
            FROM product_accounting";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                return table;
            }
        }
    }
}
```

Листинг 1.2 – SQL-запрос для выборки с условием

```
public DataTable GetRecordsByEmployee(int employeeId)
{
    if (!EmployeeExists(employeeId))
    {
        throw new ArgumentException($"Сотрудник с ID
{employeeId} не существует.");
    }

    using (var conn = _dbConnection.GetConnection())
    {
```

```
        conn.Open();
        string query = @"
            SELECT pa.productAcc_id, pa.accounting_date,
pa.quantity,
                   e.full_name AS employee_name,
                   sz.zone_name AS storage_zone
            FROM product_accounting pa
            INNER JOIN employee e ON pa.employee_id =
e.employee_id
            INNER JOIN storage_zone sz ON pa.storage_id =
sz.storage_id
            WHERE pa.employee_id = @employeeId";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("employeeId",
employeeId);
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                return table;
            }
        }
    }
}
```

Листинг 1.3 – SQL-запрос для выборки из нескольких таблиц

```
public DataTable GetAllRecords()
{
    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        string query = @"
            SELECT pa.productAcc_id, pa.accounting_date,
pa.quantity,
                   e.full_name AS employee_name,
```

```
                    sz.zone_name AS storage_zone
            FROM product_accounting pa
            INNER JOIN employee e ON pa.employee_id =
e.employee_id
            INNER JOIN storage_zone sz ON pa.storage_id =
sz.storage_id";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                return table;
            }
        }
    }
}
```

Листинг 1.4 – SQL-запрос для агрегирующей выборки

```
public DataTable GetAggregateRecords(int minRecordCount,
DateTime startDate)
{
    if (minRecordCount < 0)
    {
        throw new ArgumentException("Минимальное количество
записей не может быть отрицательным.");
    }

    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        string query = @"
            SELECT sz.zone_name AS storage_zone,
                   COUNT(pa.productAcc_id) AS record_count,
                   SUM(pa.quantity) AS total_quantity,
```

```
                    AVG(pa.quantity) AS avg_quantity
            FROM product_accounting pa
            INNER JOIN storage_zone sz ON pa.storage_id =
sz.storage_id
            WHERE pa.accounting_date >= @startDate
            GROUP BY sz.zone_name
            HAVING COUNT(pa.productAcc_id) >= @minRecordCount
            ORDER BY total_quantity DESC";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("minRecordCount",
minRecordCount);
            cmd.Parameters.AddWithValue("startDate", startDate);
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                return table;
            }
        }
    }
}
```

Листинг 1.5 – SQL-запрос для коррелированного подзапроса

```
public DataTable GetCorrelatedSubquery(int supplyId)
{
    if (!SupplyExists(supplyId))
    {
        throw new ArgumentException($"Поставка с ID {supplyId}
не существует.");
    }

    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
```

```
        string query = @"
            SELECT pa.productAcc_id, pa.accounting_date,
pa.quantity,
                   (SELECT s.supply_date FROM supply s WHERE
s.supply_id = pa.supply_id) AS supply_date,
                   (SELECT e.full_name FROM employee e WHERE
e.employee_id = pa.employee_id) AS employee_name
            FROM product_accounting pa
            WHERE pa.supply_id = @supplyId";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("supplyId", supplyId);
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                return table;
            }
        }
    }
}
```

Листинг 1.6 – SQL-запрос для некоррелированного подзапроса

```
public DataTable GetNonCorrelatedSubquery(int supplyId)
{
    if (!SupplyExists(supplyId))
    {
        throw new ArgumentException($"Поставка с ID {supplyId}
не существует.");
    }

    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        string query = @"
```

```
            SELECT pa.productAcc_id, pa.accounting_date,
pa.quantity,
                   e.full_name AS employee_name
            FROM product_accounting pa
            INNER JOIN employee e ON pa.employee_id =
e.employee_id
            WHERE pa.supply_id = @supplyId
            AND pa.quantity > (SELECT AVG(quantity) FROM
product_accounting WHERE supply_id = @supplyId)";
        using (var cmd = new NpgsqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("supplyId", supplyId);
            using (var adapter = new NpgsqlDataAdapter(cmd))
            {
                DataTable table = new DataTable();
                adapter.Fill(table);
                if (table.Rows.Count == 0)
                {
                    using (var avgCmd = new
NpgsqlCommand("SELECT AVG(quantity) FROM product_accounting
WHERE supply_id = @supplyId", conn))
                    {

avgCmd.Parameters.AddWithValue("supplyId", supplyId);
                        var avg = avgCmd.ExecuteScalar();
                        Console.WriteLine($"DEBUG: Average
quantity for supply_id {supplyId} = {avg}");
                    }
                }
                return table;
            }
        }
    }
}
```

Листинг 1.7 – SQL-запрос для операции INSERT

```
public void InsertRecord(DateTime accountingDate, int quantity,
int employeeId, int supplyId, int storageId)
{
    if (!EmployeeExists(employeeId))
    {
        throw new ArgumentException($"Сотрудник с ID
{employeeId} не существует.");
    }
    if (!SupplyExists(supplyId))
    {
        throw new ArgumentException($"Поставка с ID {supplyId}
не существует.");
    }
    if (!StorageZoneExists(storageId))
    {
        throw new ArgumentException($"Зона хранения с ID
{storageId} не существует.");
    }

    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        using (var transaction = conn.BeginTransaction())
        {
            try
            {
                string query = @"
                    INSERT INTO product_accounting
(accounting_date, quantity, employee_id, supply_id, storage_id)
                    VALUES (@accountingDate, @quantity,
@employeeId, @supplyId, @storageId)";
                using (var cmd = new NpgsqlCommand(query, conn,
transaction))
                {
```

```
cmd.Parameters.AddWithValue("accountingDate", accountingDate);
                    cmd.Parameters.AddWithValue("quantity",
quantity);
                    cmd.Parameters.AddWithValue("employeeId",
employeeId);
                    cmd.Parameters.AddWithValue("supplyId",
supplyId);
                    cmd.Parameters.AddWithValue("storageId",
storageId);
                    cmd.ExecuteNonQuery();
                }
                transaction.Commit();
            }
            catch (Exception ex)
            {
                transaction.Rollback();
                Console.WriteLine($"ERROR: Failed to insert
record. {ex.Message}");
                throw;
            }
        }
    }
}
```

Листинг 1.8 – SQL-запрос для операции UPDATE

```
public void UpdateRecord(int id, DateTime? accountingDate, int?
quantity, int? employeeId, int? supplyId, int? storageId)
{
    if (employeeId.HasValue &&
!EmployeeExists(employeeId.Value))
    {
        throw new ArgumentException($"Сотрудник с ID
{employeeId.Value} не существует.");
    }
    if (supplyId.HasValue && !SupplyExists(supplyId.Value))
    {
```

```
        throw new ArgumentException($"Поставка с ID
{supplyId.Value} не существует.");
    }
    if (storageId.HasValue &&
!StorageZoneExists(storageId.Value))
    {
        throw new ArgumentException($"Зона хранения с ID
{storageId.Value} не существует.");
    }
    if (!RecordExists(id))
    {
        throw new ArgumentException($"Запись с ID {id} не
существует.");
    }


    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        using (var transaction = conn.BeginTransaction())
        {
            try
            {
                string query = "UPDATE product_accounting SET ";
                var parameters = new List<NpgsqlParameter>();
                if (accountingDate.HasValue) { query +=
"accounting_date = @accountingDate, "; parameters.Add(new
NpgsqlParameter("accountingDate", accountingDate.Value)); }
                if (quantity.HasValue) { query += "quantity =
@quantity, "; parameters.Add(new NpgsqlParameter("quantity",
quantity.Value)); }
                if (employeeId.HasValue) { query += "employee_id
= @employeeId, "; parameters.Add(new
NpgsqlParameter("employeeId", employeeId.Value)); }
                if (supplyId.HasValue) { query += "supply_id =
@supplyId, "; parameters.Add(new NpgsqlParameter("supplyId",
supplyId.Value)); }
```

```
                if (storageId.HasValue) { query += "storage_id =
@storageId, "; parameters.Add(new NpgsqlParameter("storageId",
storageId.Value)); }

                query = query.TrimEnd(',', ' ') + " WHERE
productAcc_id = @id";

                parameters.Add(new NpgsqlParameter("id", id));


                using (var cmd = new NpgsqlCommand(query, conn,
transaction))

                {


cmd.Parameters.AddRange(parameters.ToArray());

                    cmd.ExecuteNonQuery();

                }

                transaction.Commit();

            }

            catch (Exception ex)

            {

                transaction.Rollback();

                Console.WriteLine($"ERROR: Failed to update
record. {ex.Message}");

                throw;

            }

        }

    }

}
```

Листинг 1.9 – SQL-запрос для операции DELETE

```
public void DeleteRecord(int id)

{

    if (!RecordExists(id))

    {

        throw new ArgumentException($"Запись с ID {id} не
существует.");

    }
```

```
    using (var conn = _dbConnection.GetConnection())
    {
        conn.Open();
        using (var transaction = conn.BeginTransaction())
        {
            try
            {
                string query = "DELETE FROM product_accounting
WHERE productAcc_id = @id";
                using (var cmd = new NpgsqlCommand(query, conn,
transaction))
                {
                    cmd.Parameters.AddWithValue("id", id);
                    cmd.ExecuteNonQuery();
                }
                transaction.Commit();
            }
            catch (Exception ex)
            {
                transaction.Rollback();
                Console.WriteLine($"ERROR: Failed to delete
record. {ex.Message}");
                throw;
            }
        }
    }
}
```

Листинг 1.10 – Код валидации входных данных

```
private bool ValidateDate(string input, out DateTime value,
TextBox textBox, string fieldName)
{
    value = DateTime.MinValue;
    if (string.IsNullOrEmpty(input))
    {
```

```
            textBox.BackColor = Color.White;
            toolTip.SetToolTip(textBox, $"Введите {fieldName} в
формате ДД.ММ.ГГГГ (например, 20.05.2025)");
            return false;
        }
        if (!DateTime.TryParseExact(input, "dd.MM.yyyy", null,
System.Globalization.DateTimeStyles.None, out value))
        {
            textBox.BackColor = Color.LightPink;
            toolTip.SetToolTip(textBox, $"Некорректная {fieldName}.
Введите дату в формате ДД.ММ.ГГГГ (например, 20.05.2025)!");
            return false;
        }
        textBox.BackColor = Color.White;
        toolTip.SetToolTip(textBox, $"Введите {fieldName} в формате
ДД.ММ.ГГГГ (например, 20.05.2025)");
        return true;
    }
```